Simple and Fast Implementation of Segmented Matrix Algorithm for Haar DWT on a Low Cost GPU

Madeena Sultana[1] and Nurul Muntasir Mamun[2]
[1] Dept. of Computer Science and Engineering, Jahangirnagar University, Dhaka, Bangladesh
Email: deena.sultana@gmail.com
[2] Dept. of Applied Physics Electronics and Communication Engineering, Dhaka University, Dhaka, Bangladesh
Email: mamun.muntasir@yahoo.com

***Abstract*— Haar discrete wavelet transform (DWT), the simplest among all DWTs, has diverse applications in signal and image processing fields. A traditional approach for 2D Haar DWT is 1D row operation followed by and 1D column operation. In 2002, Chen and Liao presented a fast algorithm for 2D Haar DWT based on segmented matrix. However, this method is infeasible for its high computational requirements for processing large sized images. In this paper, we have implemented the segmented matrix algorithm on a low cost NVIDIA's GPU to achieve speedup in computation. The efficiency of our GPU based implementation is measured and compared with CPU based algorithms. Our experimental results show performance improvement over a factor of 28.5 compared with Chen and Liao's CPU based segmented matrix algorithm and a factor of 8 compared to MATLAB's wavelet function for an image of size 2560×2560.**

***Index Terms*—Haar discrete wavelet transform (DWT), CUDA, GPU, segmented matrix algorithm, parallel discrete wavelet transform**

I. Background and Introduction

Discrete wavelet transforms (DWTs) has been used in a wide range of signal and image processing applications such as – image and video coding (MPEG-4 or JPEG 2000), pattern recognition, image watermarking, medical image analysis etc. In traditional approach, 2D (two-dimensional) Haar DWT is performed in two phase- one row operation, one column operation, and column operation cannot be performed until the row operation is completed. Therefore, the speed of computation degrades significantly. To address this problem, Chen and Liao [1] proposed the segmented matrix algorithm where computation is performed by data rearrangements and one matrix multiplication. Therefore, this simple algorithm can produce the same results as traditional 2D Haar DWT with a much faster speed. Moreover, it is highly suitable for parallel implementation as only two rows are involved in computation at a time.

Nowadays large size images are common due to the availability and advancement of image capturing technology. Therefore many wavelet based applications have to manage large scaled image processing. Parallel computing is a direct way of speeding up these high computation requirements. A significant amount of works have already been done for all sorts of high performance computers, for special purpose hardware [2]-[4], for FPGAs [5][6] and for SIMD architectures [7].Considerable amount of speedup is also achieved by employing GPUs with OpenGL and Cg-based implementations for DWT computations [8]-[10].

However, GPU accelerated computation became especially interesting since early 2007 when NVIDIA introduced CUDA (Compute Unified Device Architecture) enabled GPUs, which offer massive parallel computation power. Providing many hundreds of gigaflops of processing power current GPUs are leveraging the parallel computation in a more efficient way than on a CPU [11].

Being harnessed by many researches, these commodity and readily available GPUs are providing dramatic computation speedup in various research fields. Joaquín Franco, Gregorio Bernabé, Juan Fernández and Manuel E. Acacio [12] achieved significant speed up with NVIDIA's Tesla C870 over Intel's Core 2 Quad Q6700 (2.66GHz). Vaclav Simek and Ram Rakesh Asn [13] used CUDA enabled GPU for accelerated 2D wavelet based image compression. Recently, Wladimir J. van der Laan, Andrei C. Jalba and Jos B.T.M. Roerdink [14] implemented a fast hybrid method for 2D DWT on CUDA for both 2D images and 3D volume data.

In this paper we have implemented the segmented matrix algorithm for 2D Haar wavelet transform on a low cost, commodity GPU. Our objective is to achieve computation speed up to process large scaled images without increasing computational complexity and cost.

II. Traditional Computation

Haar DWT is the simplest since it only uses two low pass filter coefficients (1,1) and two high pass filter coefficients (1,-1). Haar wavelet transform in frequency domain can be obtained by addition and subtraction of the pixels of images. 2D haar DWT decomposes an input image into four sub-bands, one average component (W_{LL}) and three detail components (W_{LH}, W_{HL}, W_{HH}).

Traditionally, 2D Haar wavelet transform can be accomplished by one row and one column operations where the result of row transform is the input of column transform. Fig. 1 represents the 2D Haar wavelet transforms of a 4×4 image.

DOI: 01.IJSIP.03.01.117

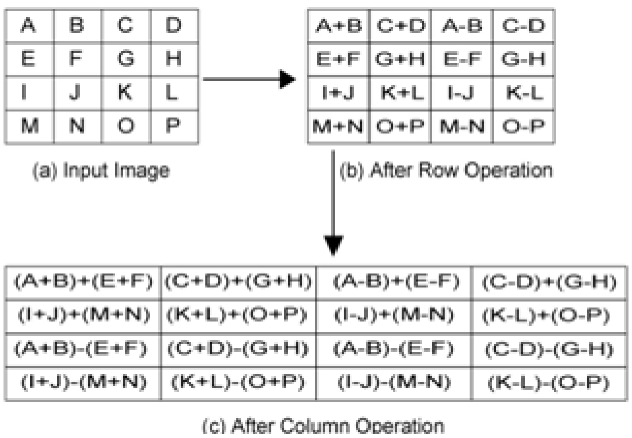


Figure.1. 2D Haar DWT of a 4×4 image by traditional approach.

III. Segmented Matrix Algorithm

Chen and Liao [1] proposed a computationally fast algorithm called "segmented matrix algorithm" where 2D Haar DWT can be performed by only one matrix multiplication instead of two separate 1D transforms. The step by step process of this algorithm is as follows.

Step 1: Consider I as the input image of size m×n. Form B_{ij}=2×2 sub-blocks from original image I where i=1…m/2 and j=1…n/2. For example,

$$B_{11} = \begin{bmatrix} I_{11} & I_{12} \\ I_{21} & I_{22} \end{bmatrix},\quad B_{12} = \begin{bmatrix} I_{13} & I_{14} \\ I_{23} & I_{24} \end{bmatrix},\ldots\quad B_{1,n/2} = \begin{bmatrix} I_{1,n-1} & I_{1n} \\ I_{2,n-1} & I_{2,n} \end{bmatrix}$$

Step 2: Z-scan each B_{ij} and generate m×n row vectors A_{ij}. For example.

$$B_{11} = \begin{bmatrix} I_{11} & I_{12} \\ I_{21} & I_{22} \end{bmatrix} = A_{11} = \begin{bmatrix} I_{11} & I_{12} & I_{21} & I_{22} \end{bmatrix}$$

Step 3: Express these row matrices as an intermediate matrix M.

$$M = \begin{bmatrix} A_{11} \\ A_{12} \\ \cdot \\ \cdot \\ A_{m/2,n/2} \end{bmatrix} = \begin{bmatrix} I_{11} & I_{12} & I_{21} & I_{22} \\ I_{13} & I_{14} & I_{23} & I_{24} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ I_{m-1,n-1} & I_{m-1,n} & I_{m,n-1} & I_{m,n} \end{bmatrix}$$

Step 4: Consider filter coefficient matrix

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Find H= M×C.

Step 5: Haar wavelet transform can be divided into four sub-matrices of size $\frac{m}{2}\times\frac{n}{2}$, $W = \begin{bmatrix} W_{LL} & W_{HL} \\ W_{LH} & W_{HH} \end{bmatrix}$

The rearrangement of the elements of H into four sub-matrices will produce the resultant Haar wavelet transform matrix W.

The rearrangements are as follows -

a) The elements in the first column of H are filled in W_{LL} row by row.

b) The elements in the second column of H are filled in W_{HL} row by row.

c) The elements in the third column of H are filled in W_{LH} row by row.

d) The elements in the fourth column of H are filled in W_{HH} row by row.

IV. CUDA Implementation

The CUDA platform is currently concentrating an enormous attention due to its tremendous potential of parallel processing. In November 2006, NVIDIA introduced CUDA with a new parallel programming model and instruction set architecture to solve many complex computational problems very efficiently [11]. Each CUDA complainant device is a set of multiprocessor cores where each core has SIMT (Single Instruction, Multiple Thread) architecture. Today four quad-core CPUs can run only 16 threads concurrently, whereas the smallest executable parallel unit on a CUDA device comprised of 32 threads. All CUDA enabled NVIDIA GPUs support at least 768 concurrently active threads per multiprocessor. Moreover, some GPUs can support 1,024 or more active threads per multiprocessor [11]. Devices comprise of 30 multiprocessors (e.g. NVIDIAGeForce GTX 280), can support more than 30,000 active threads [15]. A good parallel implementation of an application on a GPU can achieve more than 100 times speedup over sequential execution [16].

In SIMT architecture of CUDA, a portion of a parallel application executed many times independently on different data, by many threads running on different processors, at any given clock cycle. This parallel portion can be isolated into a function which is called kernel. A kernel is organized as a set of thread blocks and each thread block is, in turn, organized as a three-dimensional array of threads. Threads within the same block can efficiently cooperate through shared memory and can synchronize with each other. Each thread has its own unique thread ID which is defined by the three thread indices: threadIdx.x, threadIdx.y, and threadIdx.z. Each block is identified by a unique two-dimensional coordinate given by the CUDA specific keywords blockIdx.x and blockIdx.y. All blocks must have the equal number of threads organized exactly in the same manner. The use of multidimensional identifiers simplifies memory addressing of multidimensional data. The block and grid dimensions, collectively known as execution configuration, can be set at run-time.

In our implementation we have used blocks each having 16×16 threads. The grid size is set at run-time according to the size of input image. Our CUDA implementation consists of the following steps:

1. Copy image data from host memory to GPU memory.
2. Determine the execution configuration.
3. GPU executes kernel to compute the elements of the intermediate matrix M on each core in a parallel fashion.

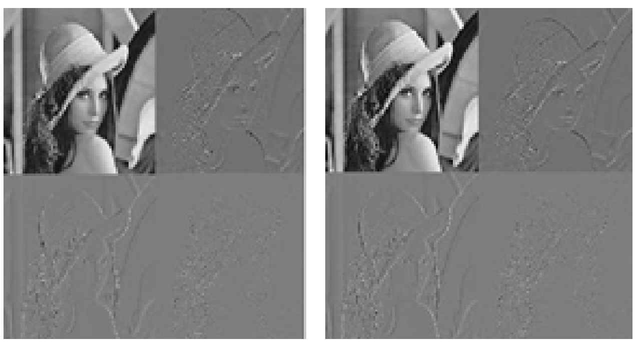
DOI: 01.IJSIP.03.01.117

4. The resulted matrix H is computed simultaneously in GPU.
5. Copy the result from GPU memory to host memory.

The CPU-based algorithm is implemented on Intel Pentium IV, 3.00GHz processor equipped with 512MB DDR2 RAM. The GPU based algorithm is tested on NVIDIA GeForce 8500GT graphics card containing 16 cores, maximum 512 threads per block and 512 MB global memory.

V. Results and Discussion

To test the computational efficiency of our GPU based segmented matrix algorithm, we have taken images of different sizes as inputs. Fig. 2 shows one level 2D Haar DWT of 256×256 lena image using CPU based and GPU based segmented matrix algorithm. For comparison we also have considered MATLAB's dwt2() function from wavelet toolbox and the CPU implementation of segmented matrix algorithm.



(a) (b)

Figure.2. One level 2D Haar DWT using (a) CPU based and (b) GPU based segmented matrix algorithm.

Table I represents the comparison of computing time of MATLAB's dwt2(), segmented matrix algorithm on CPU and on GPU with increasing size of input images.

TABLE.I. Computation Time Comparison Relative to Image Size

Image Size	Matlab's dwt2() on CPU (in seconds)	Segmented Matrix Algorithm on CPU (in seconds)	Segmented Matrix Algorithm on GPU (in seconds)
32×32	0.252586	0.001253	0.082350
64×64	0.255753	0.001728	0.083025
128×128	0.273688	0.004898	0.085409
256×256	0.279187	0.028654	0.085688
512×512	0.351286	0.094345	0.094245
1024×1024	0.662187	1.379558	0.118496
1536×1536	1.692166	11.262825	0.555008
2048×2048	2.903447	27.884286	0.888735
2560×2560	18.328886	65.466872	2.293685

Table I shows that the performance of CPU based segmented matrix algorithm declined noticeably for large sized images, although it performed better for small sized images. In contrast, GPU based implementation of this algorithm improved the performance for large over a factor of 10 to 28 for images sized 1024×1024 to 2560×2560. Moreover, it performed better than MATLAB's wavelet function for all small and large sized images. Therefore, among the three algorithms our GPU based segmented matrix algorithm performed the best for high resolution images.

However, the main drawback of GPU computation is the transfer time between the host memory and device memory. The time needed to copy data from the host's memory to GPU's global memory requires a large fraction of total execution time. Therefore, if we exclude the data transfer time from execution time, we would get significant speedup for large sized images.

Conclusions

The widespread usage of the Haar Discrete Wavelet Transform (DWT) has motivated the implementation of a simple and low cost GPU based DWT algorithm. Our experimental results show that for an image of size 2560×2560, the GPU based segmented matrix algorithm is more than 28.5 times faster than CPU computation including data transfer. Moreover, this GPU based method achieved approximately 8x speedup than the CPU based computation of MATLAB's dwt2() for the same image. Due to the speedy calculations we believe that the ideas presented in this paper will have widespread applications in processing large sized images.

References

[1] P. Y. Chen and E. C. Liao, "A new algorithm for Haar discrete wavelet transform," *IEEE International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 453-457, 2002.

[2] M. Martina, G.Masera, G.Piccinini and M.Zamboni, "A VLSI Architecture for IWT (Integer Wavelet Transform)," *Proc. of 43rd Midwest Symposium on Circuits and Systems*, pp. 1174-1177, August 2000.

[3]K. Haapala, P. Kolinummi, T. Hamalainen, and J. Saarinen, "Parallel DSP implementation of wavelet transform in image compression," *Proc. of ISCAS IEEE International Symposium on Circuits and Systems*, vol. 5. pp. 89-92, 2000.

[4] Matthias Hopf and Thomas Ertl., "Hardware Accelerated Wavelet Transformations," *Proc. of EG/IEEE TCVG Symposium on Visualization VisSym*, pp. 93–103, 2000.

[5] C. Graves and C. Gloster, "Use of dynamically reconfigurable logic in adaptive wavelet packet applications," *Proc. of the 5th Canadian Workshop on Field-Programmable Devices*, June 1998.

[6] Baofeng Li, Yong Dou, Haifang Zhou, and Xingming Zhou, "FPGA accelerator for wavelet-based automated global image registration," *EURASIP J. Embedded Syst.*, pp. 1–10, 2009.

[7] Mats Holmström, "Parallelizing the fast wavelet transform," *Parallel Computing*, vol. 11(21), pp. 837-1848, April 1995.

[8] T. T. Wong, C. S. Leung, P. A. Heng, and J. Wang, "Discrete wavelet transform on consumer-level graphics hardware," *IEEE Transactions on Multimedia*, vol. 9(3), pp. 668–673, April 2007.

DOI: 01.IJSIP.03.01.117

ACEEE

[9] C. Tenllado, J. Setoain, M. Prieto, L. Piñuel, and F. Tirado, "Parallel Implementation of the 2D Discrete Wavelet Transform on Graphics Processing Units: Filter Bank versus Lifting," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19(3), pp. 299-310, 2008.

[10] Antonio Garcia and Han-Wei Shen, "GPU-based 3D wavelet reconstruction with tileboarding," *The Visual Computer* , vol. 21(8), pp. 755-763, September 2005.

[11] "NVIDIA CUDA C Programming Guide 4.0," Available at http://developer.nvidia.com/cuda-toolkit-3.1-downloads, accessed August 02, 2011.

[12] Joaquín Franco, Gregorio Bernabé, Juan Fernández, and Manuel E. Acacio, "A Parallel Implementation of the 2D Wavelet Transform Using CUDA," *Proc. of International Conf. on Parallel, Distributed and Network-based Processing*, pp.111-118, 2009.

[13] Vaclav Simek and Ram Rakesh Asn, "GPU Acceleration of 2D-DWT Image Compression in MATLAB with CUDA," *Second UKSIM European Symposium on Computer Modeling and Simulation*, pp.274-277, 2008.

[14] Wladimir J. van der Laan, Andrei C. Jalba, and Jos B.T.M. Roerdink, "Accelerating Wavelet Lifting on Graphics Hardware Using CUDA," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22(1), pp. 132-146, January 2011.

[15] "CUDA C best practices guide," Available at http://developer.nvidia.com/cuda-toolkit-31-downloads, accessed August 05, 2011.

[16] David B. Kirk and Wen-mei W. Hwu, *Programming massively parallel processors- a hands-on approach*, Elsevier Inc., USA, January 22, 2010.

DOI: 01.IJSIP.03.01.117